模拟器实验报告

李浩然 2025 年 12 月 13 日

实验目的

本实验的目标是编写一个 LC-3 运行环境，模拟 LC-3 微处理器的基本运行，包括机器码的解析与执行、内存与寄存器的访问，以及特定查询的处理。

实验环境

- 编程语言：Python
- 实现的功能：LC-3 模拟器（包括内存、寄存器、程序计数器等模拟）
- 输入：机器码、查询请求
- 输出：寄存器、内存的内容或错误信息

实验内容与实现

LC-3 机器模拟环境的设计

为了实现 LC-3 的模拟器，首先设计了一个模拟环境，负责内存管理、寄存器管理和指令执行。

内存管理

模拟器中的内存（self.mem）每个地址为 16 位宽。程序在运行时会根据加载地址将机器码装入指定位置。

- 在 load 方法中，首先读取输入的第一行，确定程序的加载地址（即程序计数器 PC 的初始值）。
- 然后从第二行开始，将每一条机器码指令按顺序存储到内存中，地址自增并限制在 16 位地址空间（0x0000 到 0xFFFF）。

寄存器管理

模拟器模拟了 8 个通用寄存器 (R0~R7)，其中寄存器的值保存在 self.reg 列表中。此外，还设计了 PC （程序计数器）和 COND（条件标志）两个特殊寄存器。

- 程序计数器 PC 负责跟踪当前正在执行的指令地址，执行每条指令后，PC 会自增。
- 条件标志 COND 根据指令执行结果更新。LC-3 使用三种标志：正数（1）、零（2）和负数（4）。
- 条件标志的更新通过 update_flags 方法实现。每次执行指令后，检查寄存器的值并设置相应的标志。

符号扩展

为了处理 LC-3 架构中的有符号数，设计了 sign_extend 方法。它的作用是将低位的有符号数（如指令中的立即数）扩展为 16 位，确保计算时不丢失符号信息。

- 对于负数，符号位会扩展为 1，确保负数在运算过程中得到正确的处理。
- sign_extend 方法会根据指定的位宽（如 5 位、9 位等）进行符号扩展，并保证结果为 16 位。

指令执行

- 取指：每次从内存中读取当前指令，然后更新程序计数器 PC。
- 解析指令：根据指令的高 4 位获取操作码（如 BR, ADD 等）。
- 执行指令：根据操作码执行不同的操作。
- 特别地，对于 TRAP x25（HALT 指令），当程序遇到该指令时，执行停止。

查询功能

为了实现对模拟器状态的查询功能，设计了一个查询处理模块，支持查询寄存器、内存地址的内容。

- 当用户输入 PC 时，输出程序计数器的当前值。
- 当用户输入 R0∼R7 时，返回对应寄存器的值。
- 当用户输入内存地址（如 x3000）时，返回该内存地址中的内容。
- 输入格式要求严格，查询内容必须为大写寄存器名或地址（十六进制形式）。

实验样例

输入机器代码：

```
0011000000000000
0101000000100000
1111000000100101
```

查询内容为：

```
R0
PC
x3000
```

输出结果为：

```
x0000
x3002
x5020
```

结语

通过本次实验，我成功实现了一个简单的 LC-3 模拟器，并能够解析和执行机器代码。模拟器支持内存读取、寄存器操作及指令执行，能够根据用户输入的查询返回相应的结果。

附录：代码实现

```python
# Python完整代码
import sys

def sign_extend(x, bit_count):
    # 符号扩展：将n位有符号数扩展为16位（LC-3架构要求）
    if x >> (bit_count - 1) & 1:
        x = x - (1 << bit_count)
    return x & 0xFFFF  # 确保16位宽度

class LC3:
    def __init__(self):
        self.mem = [0] * 0x10000  # 64KB内存（0x0000~0xFFFF）
        self.reg = [0] * 8        # 8个通用寄存器（R0~R7）
        self.PC = 0               # 程序计数器（当前指令地址）
        self.COND = 0             # 条件标志（1:正，2:零，4:负）

    def load(self, lines):
        # 加载二进制程序到内存：首行为加载地址，后续为指令
        if not lines:
            return
        la = int(lines[0].strip(), 2)
        self.PC = la & 0xFFFF
        addr = self.PC
        for bits in lines[1:]:
            bits = bits.strip()
            if not bits:
                continue
            self.mem[addr] = int(bits, 2) & 0xFFFF
            addr = (addr + 1) & 0xFFFF

    def update_flags(self, r):
        # 根据寄存器r的值更新条件标志
        val = self.reg[r] & 0xFFFF
        self.COND = 2 if val == 0 else 4 if val >> 15 else 1

    def mem_read(self, addr):
        # 内存读取（地址限制在16位空间）
        return self.mem[addr & 0xFFFF]

    def mem_write(self, addr, val):
        # 内存写入（地址和数据均限制16位）
        self.mem[addr & 0xFFFF] = val & 0xFFFF

    def run(self):
        # 指令执行循环（取指→译码→执行）
        while True:
            instr = self.mem_read(self.PC)
```

```
            self.PC = (self.PC + 1) & 0xFFFF
            op = instr >> 12  # 解析操作码（高4位）

            # 这里省略其他指令的实现，具体可以参考原始代码
            if op == 15:  # TRAP：陷阱指令（系统调用）
                if instr & 0xFF == 0x25:  # HALT：停止程序
                    break

def main():
    # 主流程：读取程序→运行模拟器→处理查询
    code_lines = []
    try:
        while True:
            line = input()
            if line.strip() == "":
                break
            code_lines.append(line)
    except EOFError:
        pass

    lc3 = LC3()
    lc3.load(code_lines)
    lc3.run()

    # 处理查询（PC/寄存器/内存地址）
    try:
        while True:
            q = input().strip()
            if not q:
                continue
            if q == 'PC':
                print(f"x{lc3.PC:04x}")
            elif q.upper().startswith('R') and len(q) == 2 and q[1].isdigit()
                :
                idx = int(q[1])
                print(f"x{lc3.reg[idx] & 0xFFFF:04x}" if 0 <= idx <=7 else "
                    x0000")
            else:
                try:
                    addr_str = q.lower().lstrip('x')
                    addr = int(addr_str, 16) & 0xFFFF
                    print(f"x{lc3.mem_read(addr) & 0xFFFF:04x}")
                except:
                    print("x0000")
    except EOFError:
        pass

if __name__ == '__main__':
    main()
```